

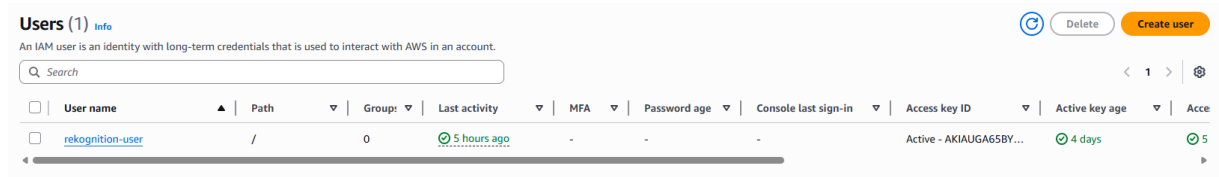
Proje: Video Akışı ve İşleme Uygulaması

Projenin Amacı: Bu projede, gerçek zamanlı video akışlarının yönetilmesi ve bu akışlar üzerinden nesne tanıma, etiketleme gibi analizlerin yapılması amaçlanmıştır. Bulut tabanlı video işleme çözümleri kullanılarak, esnek ve ölçeklenebilir bir sistem geliştirilmiştir.

GELİŞTİRME SÜRECİ

Kurulumlar

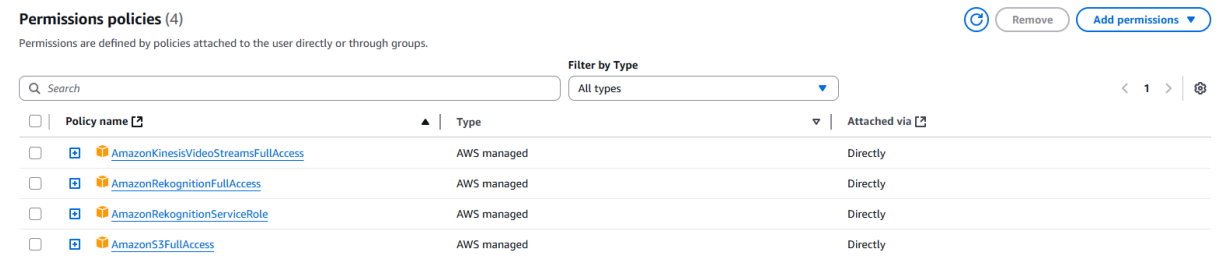
AWS hesabı açıldı ve IAM (Identity and Access Management) bölümünden Users kısmından kullanıcı oluşturuldu .(rekognition-user)



The screenshot shows the AWS IAM console 'Users' page. It lists one user named 'rekognition-user'. The user is active, has no MFA, and their password expires in 5 hours. They have an access key that is active for 4 days.

User name	Path	Group	Last activity	MFA	Password age	Console last sign-in	Access key ID	Active key age	Access key status
rekognition-user	/	0	5 hours ago	-	-	-	Active - AKIAUGA65BY...	4 days	5

AmazonKinesisVideoStreamsFullAccess, AmazonRekognitionFullAccess, AmzmonRekognitionServiceRole, AmazonS3FullAccess izinleri verildi.

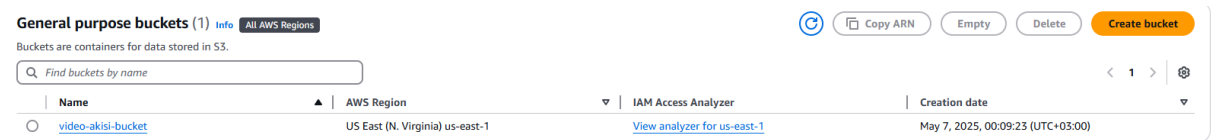


The screenshot shows the AWS IAM console 'Permissions policies' page. It lists four policies attached to the user: AmazonKinesisVideoStreamsFullAccess, AmazonRekognitionFullAccess, AmazonRekognitionServiceRole, and AmazonS3FullAccess. All policies are AWS managed and attached directly.

Policy name	Type	Attached via
AmazonKinesisVideoStreamsFullAccess	AWS managed	Directly
AmazonRekognitionFullAccess	AWS managed	Directly
AmazonRekognitionServiceRole	AWS managed	Directly
AmazonS3FullAccess	AWS managed	Directly

Access key, Secret Access key ve Region bilgileri alındı.

Video dosyalarının geçici olarak saklanması ve analiz edilebilmesi amacıyla bir S3 Bucket oluşturuldu. (video-akisi-bucket)



The screenshot shows the AWS S3 console 'General purpose buckets' page. It lists one bucket named 'video-akisi-bucket'. The bucket is located in the US East (N. Virginia) region and was created on May 7, 2025.

Name	AWS Region	IAM Access Analyzer	Creation date
video-akisi-bucket	US East (N. Virginia) us-east-1	View analyzer for us-east-1	May 7, 2025, 00:09:23 (UTC+03:00)

MongoDB Atlas hesabı açıldı. Proje oluşturuldu. Cluster oluşturuldu. (Cluster 0)

Overview

Connect kısmına girip, ardından Drivers kısmına girip Cluster'a bağlanabilmek için URI alındı.

```
mongodb+srv://<db_username>:<db_password>@cluster0.18atnrk.mongodb.net/?
retryWrites=true&w=majority&appName=Cluster0
```

Collection ve Database oluşturuldu. Verilerimiz buraya kaydedilecek.

Kodlar

AWS S3'e Video Yükleme (S3_uploader.py)

Projede video dosyalarının bulutta saklanması amacıyla AWS S3 (Simple Storage Service) kullanılmıştır. Python programlama diliyle birlikte boto3 kütüphanesi aracılığıyla, yerel dosyaların doğrudan S3 Bucket içerisine yüklenmesi sağlanmıştır.

boto3.client('s3', ...) komutu ile S3 servisine erişim sağlanmıştır.

AWS hesabı üzerinden alınan Access Key, Secret Access Key ve Region bilgileri kullanılarak kimlik doğrulama yapılmıştır.

upload_file fonksiyonu, belirtilen file_path'teki yerel dosyayı bucket_name adlı S3 klasörüne, s3_key ile tanımlanan konuma yüklemektedir.

İşlem başarıyla tamamlandığında kullanıcıya bilgi veren bir çıktı mesajı yazdırılmıştır. Bu sayede, video analizinden önce veya sonra içerik güvenli bir şekilde bulut ortamında saklanabilmektedir.

```

s3_uploader.py > ...
1  import boto3
2
3  def upload_video_to_s3(file_path, bucket_name, s3_key):
4      import boto3
5
6      # Kimlik bilgilerini manuel olarak ayarlamak
7      s3 = boto3.client('s3',
8                          aws_access_key_id='AKIAUGA65BY340A7AVX2',
9                          aws_secret_access_key='oHEKrgwzKSumdVMVCA3XSy4AY/4npu9jIMZt5BcY',
10                         region_name='us-east-1')
11
12     s3.upload_file(file_path, bucket_name, s3_key)
13     print(f"{file_path} dosyası {bucket_name}/{s3_key} olarak yüklendi.")
14

```

MongoDB ile Video Analiz Sonuçlarının Saklanması (mongo_handler.py)

Projede video analizlerine ait çıktıların saklanması ve gerektiğinde tekrar erişilebilmesi amacıyla MongoDB Atlas kullanılmıştır. Python ile veritabanı bağlantısı kurmak için pymongo kütüphanesi tercih edilmiştir.

MongoDBHandler sınıfı, video analiz sonuçlarının MongoDB'ye kaydedilmesi, sorgulanması ve bağlantının güvenli bir şekilde kapatılması işlemlerini gerçekleştirmektedir.

__init__ fonksiyonu ile MongoDB Atlas üzerinde oluşturulan proje12 adlı veritabanına bağlantı sağlanmaktadır.

Bağlantı için MongoClient kullanılmış ve bağlantı URI'si doğrudan sınıf içinde tanımlanmıştır.

```

mongo_handler.py > ...
1  from pymongo import MongoClient
2  from datetime import datetime
3  import json
4
5  class MongoDBHandler:
6      def __init__(self):
7          # Doğrudan bağlantı bilgilerini burada tanımlayın
8          self.MONGODB_URI = "mongodb+srv://user11:test11@cluster0.18atnrk.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0"
9          self.DB_NAME = "proje12"
10         self.COLLECTION_NAME = "proje12"
11
12         self.client = MongoClient(self.MONGODB_URI)
13         self.db = self.client[self.DB_NAME]
14

```

save_analysis_results fonksiyonu, analiz edilen videoya ait JSON dosyasını okuyarak içeriğini MongoDB'ye kaydeder. Kayıt sırasında videonun adı, analiz tarihi ve etiket verileri (labels) birlikte tutulur.

get_results fonksiyonu ile daha önce veritabanına kaydedilmiş analiz sonuçları istenirse video adına göre, istenirse tümüyle sorgulanabilir.

close_connection fonksiyonu ile veritabanı bağlantısının sistem kaynaklarını gereksiz yere tüketmemesi için güvenli bir şekilde kapatılması sağlanır.

```

def save_analysis_results(self, video_name, json_path):
    """Video analiz sonuçlarını MongoDB'ye kaydeder"""
    collection = self.db[self.COLLECTION_NAME]

    with open(json_path) as f:
        labels = json.load(f)

    document = {
        "video_name": video_name,
        "analysis_date": datetime.now(),
        "labels": labels,
        "status": "completed"
    }

    return collection.insert_one(document).inserted_id

def get_results(self, video_name=None):
    """Sonuçları sorgular"""
    collection = self.db[self.COLLECTION_NAME]
    if video_name:
        return collection.find_one({"video_name": video_name})
    return list(collection.find({}))

def close_connection(self):
    """Bağlantıyı güvenli şekilde kapat"""
    if hasattr(self, 'client') and self.client:
        self.client.close()
        self.client = None
        self.db = None

```

AWS Rekognition ile Video Etiketleme (Label Detection) (rekognition_handler.py)

AWS Rekognition servisi kullanılarak S3'e yüklenen videolar üzerinde nesne tanıma (label detection) işlemleri gerçekleştirilmiştir. Python ve boto3 kütüphanesi aracılığıyla, videonun işlenmesi, etiketlerin alınması ve çıktının dosyaya kaydedilmesi sağlanmıştır.

AWS erişim bilgileri (Access Key, Secret Key, Region) kullanılarak Rekognition servisine bağlantı kurulmuştur.

```

rekognition_handler.py > ...
1  import boto3
2  import time
3  import json
4
5  # 📌 Buraya kendi bilgilerini yaz:
6  AWS_ACCESS_KEY = 'AKIAUGA65BY340A7AVX2'
7  AWS_SECRET_KEY = 'oHEKrgwzKSumdVMVCA3XSy4AY/4npu9jIMZt5BcY'
8  AWS_REGION = 'us-east-1' # veya kendi bölgen ne ise
9
10 # Rekognition client'ını oluştur
11 rekognition = boto3.client(
12     'rekognition',
13     aws_access_key_id=AWS_ACCESS_KEY,
14     aws_secret_access_key=AWS_SECRET_KEY,
15     region_name=AWS_REGION
16 )
17

```

start_label_detection fonksiyonu, belirtilen S3 bucket içerisindeki bir video dosyasında etiket algılama işlemini başlatır. AWS tarafından bu işlem için bir JobId döndürülür. Bu ID, ilerleyen aşamada sonucu sorgulamak için gereklidir.

get_label_detection_result fonksiyonu, JobId yardımıyla etiket algılama işleminin tamamlanıp tamamlanmadığını periyodik olarak kontrol eder. İşlem başarıyla tamamlandığında sonuçları

içeren veri döndürülür. Eğer işlem başarısız olursa bir hata gösterilir, işlem devam ediyorsa 5 saniyede bir tekrar kontrol edilir.

save_result_to_file fonksiyonu ile elde edilen sonuçlar bir JSON dosyasına kaydedilir.

```
def start_label_detection(bucket_name, video_name):
    response = rekognition.start_label_detection(
        Video={'S3Object': {'Bucket': bucket_name, 'Name': video_name}}
    )
    return response['JobId']

def get_label_detection_result(job_id):
    print("AWS Rekognition sonucu bekleniyor...")
    while True:
        result = rekognition.get_label_detection(JobId=job_id)
        status = result['JobStatus']
        if status == 'SUCCEEDED':
            print("Etiketleme tamamlandı ✅")
            return result
        elif status in ['FAILED', 'ERROR']:
            raise Exception(f"İşlem başarısız: {status}")
        else:
            print("Bekleniyor... İşlem devam ediyor...")
            time.sleep(5)

def save_result_to_file(result, filename):
    with open(filename, 'w') as f:
        json.dump(result, f, indent=4)
    print(f"Etiketler {filename} dosyasına kaydedildi.")
```

Etiketli Video Görselleştirme Modülü (video_drawer.py)

AWS Rekognition tarafından analiz edilmiş ve etiketlenmiş bir videonun üzerine nesne sınırlayıcı kutularını (bounding box) ve etiket adlarını çizerek videoyu kullanıcıya görsel olarak sunar. Python'da OpenCV ve JSON verisi kullanılarak gerçekleştirilmiştir.

Video ve JSON Etiket Dosyasını Okuma: Verilen video_path adresindeki video okunur. AWS Rekognition tarafından üretilmiş JSON dosyası açılır ve içindeki etiket verileri yüklenir.

Etiketleri Timestamp'e Göre Gruplama: AWS etiketi her Timestamp (zaman damgası) için ayrı şekilde gruplandırılmıştır. Her zaman noktasındaki etiketler, bir sözlükte saklanır. Böylece her frame ile doğru etiket eşleşmesi yapılabilir.

```

video_drawer.py > ...
1  import cv2
2  import json
3
4  def draw_labels_on_video(video_path, json_path):
5      # Video yakalayıcıyı başlat
6      video_capture = cv2.VideoCapture(video_path)
7      if not video_capture.isOpened():
8          raise ValueError("Video açılmadı")
9
10     with open(json_path, 'r') as f:
11         data = json.load(f)
12
13     # Video özelliklerini al
14     fps = video_capture.get(cv2.CAP_PROP_FPS)
15     width = int(video_capture.get(cv2.CAP_PROP_FRAME_WIDTH))
16     height = int(video_capture.get(cv2.CAP_PROP_FRAME_HEIGHT))
17
18     # Etiketleri timestamp'e göre grupla
19     labels_by_timestamp = {}
20     for item in data['Labels']:
21         timestamp = item.get('Timestamp', 0)
22         label_info = item.get('Label', {})
23         name = label_info.get('Name', 'Unknown')
24         instances = label_info.get('Instances', [])
25         if timestamp not in labels_by_timestamp:
26             labels_by_timestamp[timestamp] = []
27             labels_by_timestamp[timestamp].append({
28                 'name': name,
29                 'instances': instances
30             })
31
32     current_labels = [] # En sonki aktif etiketler
33

```

Videoyu Kare Kare Okuma ve Etiketleri Çizme: Video, kare kare okunur. Her karenin zaman damgası alınır ve varsa o zamana ait etiketler kareye çizilir. Her etiketin bounding box bilgisi varsa kutu çizilir, yoksa sadece metin olarak etiketi gösterilir.

Görüntüyü JPEG Formatında Yayınlama: Her frame JPEG formatına dönüştürülerek bir web arayüzünde ya da canlı yayında kullanılabilir hâle getirilmiştir. yield komutu sayesinde video canlı olarak gösterilebilir.

```

while True:
    ret, frame = video_capture.read()
    if not ret:
        break

    current_time_ms = video_capture.get(cv2.CAP_PROP_POS_MSEC)
    current_timestamp = int(current_time_ms)

    # Eğer yeni bir timestamp varsa güncelle
    if current_timestamp in labels_by_timestamp:
        current_labels = labels_by_timestamp[current_timestamp]

    # Mevcut aktif etiketleri çiz
    y_offset = 30
    for label in current_labels:
        name = label['name']
        instances = label['instances']
        if instances:
            for inst in instances:
                box = inst.get('BoundingBox', None)
                if box:
                    x1 = int(box['Left'] * width)
                    y1 = int(box['Top'] * height)
                    x2 = x1 + int(box['Width'] * width)
                    y2 = y1 + int(box['Height'] * height)

                    cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
                    cv2.putText(frame, name, (x1, y1 - 10),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)
                else:
                    cv2.putText(frame, name, (10, y_offset),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 255), 2)
                    y_offset += 30

    # Frame'i JPEG'e dönüştür
    ret, jpeg = cv2.imencode('.jpg', frame)
    if not ret:
        break
    yield (b'--frame\r\n'
           b'Content-Type: image/jpeg\r\n\r\n' + jpeg.tobytes() + b'\r\n\r\n\r\n')

video_capture.release()

```

Proje Arayüzü (index.html)

Kullanıcıdan bir video dosyası seçmesi istenir, ardından video sunucuya yüklenir ve işlendikten sonra görsel çıktısı arayüzde gösterilir. Sayfa Türkçe dilinde hazırlanmıştır.

HTML Yapısı:

- `<h2>AWS Rekognition Video Etiketleyici</h2>`: Sayfa başlığıdır. AWS Rekognition hizmetiyle video işleme yapılacağı belirtilmiştir.
- `<input type="file">`: Kullanıcının video dosyası seçmesi için bir alan sunar.
- `<button onclick="upload()">Gönder ve İşle</button>`: Seçilen videonun sunucuya gönderilmesini ve işlenmesini başlatır.
- `<p id="status"></p>`: Kullanıcıya yükleme durumu, hata ya da başarı mesajı göstermek için kullanılır.
- `<div id="videoContainer">`: İşlenen videonun görsel çıktısını göstermek için oluşturulmuştur.

CSS Stili:

- Renk değişkenleri `:root` içinde tanımlanmıştır. Bu, sayfa genelinde tutarlı renk kullanımı sağlar.
- `.container`, `.upload-section`, ve `#videoStream` gibi sınıflarla kullanıcı arayüzü görsel olarak şık ve responsive (mobil uyumlu) hale getirilmiştir.
- Butonlar ve dosya seçici için hover ve aktif efektler ile kullanıcı deneyimi geliştirilmiştir.
- `@media (max-width: 768px)` kısmıyla mobil cihazlara özel düzenlemeler yapılmıştır.

JavaScript Fonksiyonu (upload):

Kodun sonunda yer alan JavaScript kısmı, video dosyasının yüklenmesi ve işlenmesi sürecini yönetir:

- upload() fonksiyonu çağrıldığında önce dosya seçilip seçilmediği kontrol edilir.
- Dosya varsa FormData nesnesine eklenir ve /upload adresine POST isteği gönderilir.
- Sunucudan başarı yanıtı alınırsa videoStream alanına /video_feed adresinden gelen görsel veri gösterilir.
- İşlem sırasında ve sonrasında kullanıcıya bilgi mesajları gösterilir.
- Hata durumunda kullanıcı uyarılır.

Sunucu Tarafı:

- /upload: Backend (Flask) bu POST isteğini karşılayıp videoyu işler.
- /video_feed: İşlenen videonun çıktısını sürekli olarak (örneğin bir etiketleme sonucu) sağlayan bir akış URL'sidir.

Main kod (app.py)

Python tabanlı web uygulaması, Flask framework'ü kullanılarak geliştirilmiş ve bir video dosyasının Amazon S3'e yüklenmesi, AWS Rekognition servisi ile etiketlenmesi, bu etiketlerin MongoDB'ye kaydedilmesi ve sonuçların görsel olarak kullanıcıya sunulması işlemlerini gerçekleştirmektedir.

Gerekli modüller ve yardımcı dosyalar içe aktarılır. mongo_handler, s3_uploader, rekognition_handler, video_drawer gibi dosyalar özel görevler için modülerleştirilmiş durumdadır.

```
app.py > ...
1 from flask import Flask, request, jsonify, render_template, Response
2 from flask_cors import CORS
3 import os
4 from mongo_handler import MongoDBHandler
5 from s3_uploader import upload_video_to_s3
6 from rekognition_handler import start_label_detection, get_label_detection_result, save_result_to_file
7 from video_drawer import draw_labels_on_video
8 import atexit
9 import signal
10 import sys
```

Flask uygulaması oluşturulur. CORS aktif edilerek farklı portlardan gelen istekler kabul edilir.

Video dosyalarının yükleneceği uploads klasörü tanımlanır ve yoksa oluşturulur.

MongoDB ile bağlantı kurmak için MongoDBHandler sınıfından bir nesne oluşturulur.

Uygulamada geçici olarak kullanılacak video ve etiket dosya yolları current_video_path ve current_json_path de tutulur.


```

app = Flask(__name__)
CORS(app)

# Yapılandırma
UPLOAD_FOLDER = 'uploads'
os.makedirs(UPLOAD_FOLDER, exist_ok=True)

# MongoDB bağlantısı
mongo_handler = MongoDBHandler()

# Global değişkenler
current_video_path = None
current_json_path = None

```

Kaynak temizlenir.

```

def cleanup(signum=None, frame=None):
    """Kaynakları temizle ve bağlantıyı kapat"""
    try:
        if mongo_handler:
            mongo_handler.close_connection()
            print("\nKaynaklar temizlendi, uygulama kapatılıyor...")
    except Exception as e:
        print(f"Temizleme sırasında hata: {e}")

    if signum is not None:
        sys.exit(0)

```

Uygulama kapatılırken MongoDB bağlantısı düzgün bir şekilde sonlandırılır.

Ctrl+C (SIGINT) veya sistemden gelen sonlandırma sinyalleri (SIGTERM) yakalanarak cleanup fonksiyonu çalıştırılır.

```

# Çıkış sinyallerini yakala
atexit.register(cleanup)
signal.signal(signal.SIGINT, cleanup)
signal.signal(signal.SIGTERM, cleanup)

```

Ana sayfa için index.html şablonu döndürülür.

```

@app.route('/')
def index():
    return render_template('index.html')

```

Kullanıcı video yüklediğinde bu endpoint çalışır.

Global değişkenler güncelleneceği için global olarak tanımlanır.

```
@app.route('/upload', methods=['POST'])
def upload():
    global current_video_path, current_json_path
```

Formda 'video' alanı yoksa hata döndürülür.

Dosya seçilmediyse hata döndürülür.

```
if 'video' not in request.files:
    return jsonify({"error": "Video dosyası bulunamadı"}), 400

file = request.files['video']
if file.filename == '':
    return jsonify({"error": "Dosya seçilmedi"}), 400
```

Dosya uploads klasörüne kaydedilir.

AWS işlemleri için gerekli dosya adları belirlenir.

JSON dosyası MongoDB'ye kaydedilir.

Global değişkenler güncellenir.

Kullanıcıya başarılı yanıt ve veritabanı ID'si döndürülür.

```
try:
    video_filename = file.filename
    video_path = os.path.join(UPLOAD_FOLDER, video_filename)
    file.save(video_path)

    # AWS işlemleri
    bucket_name = 'video-akisi-bucket'
    s3_key = video_filename
    json_filename = os.path.splitext(video_filename)[0] + '_labels.json'
    json_path = os.path.join(UPLOAD_FOLDER, json_filename)

    upload_video_to_s3(video_path, bucket_name, s3_key)
    job_id = start_label_detection(bucket_name, s3_key)
    result = get_label_detection_result(job_id)
    save_result_to_file(result, json_path)

    # MongoDB'ye kaydet
    mongo_id = mongo_handler.save_analysis_results(video_filename, json_path)

    # Global değişkenleri güncelle
    current_video_path = video_path
    current_json_path = json_path

    return jsonify({
        "success": True,
        "message": "İşlem başarıyla tamamlandı",
        "mongo_id": str(mongo_id)
    })
```

```

except Exception as e:
    return jsonify({
        "error": str(e),
        "message": "İşlem sırasında hata oluştu"
    }), 500

```

Önce video ve JSON dosyasının yüklü olup olmadığı kontrol edilir.

OpenCV kullanarak video üzerine etiketler çizilir ve tarayıcıya canlı akış yapılır.

```

@app.route('/video_feed')
def video_feed():
    global current_video_path, current_json_path

    if not current_video_path or not current_json_path:
        return jsonify({"error": "Video veya etiket dosyası bulunamadı"}), 404

    try:
        return Response(
            draw_labels_on_video(current_video_path, current_json_path),
            mimetype='multipart/x-mixed-replace; boundary=frame'
        )
    except Exception as e:
        return jsonify({"error": str(e)}), 500

```

MongoDB'den tüm analiz sonuçları getirilir.

```

@app.route('/results')
def get_all_results():
    try:
        results = mongo_handler.get_results()
        return jsonify({
            "success": True,
            "results": results
        })
    except Exception as e:
        return jsonify({"error": str(e)}), 500

```

Verilen video_name parametresi ile MongoDB'den tek bir analiz sonucu döndürülür.

```

@app.route('/results/<video_name>')
def get_single_result(video_name):
    try:
        result = mongo_handler.get_results(video_name=video_name)
        if result:
            return jsonify({
                "success": True,
                "result": result
            })
        return jsonify({
            "success": False,
            "message": "Sonuç bulunamadı"
        }), 404
    except Exception as e:
        return jsonify({"error": str(e)}), 500

```

Flask sunucusu başlatılır.

Hata oluşursa loglanır.

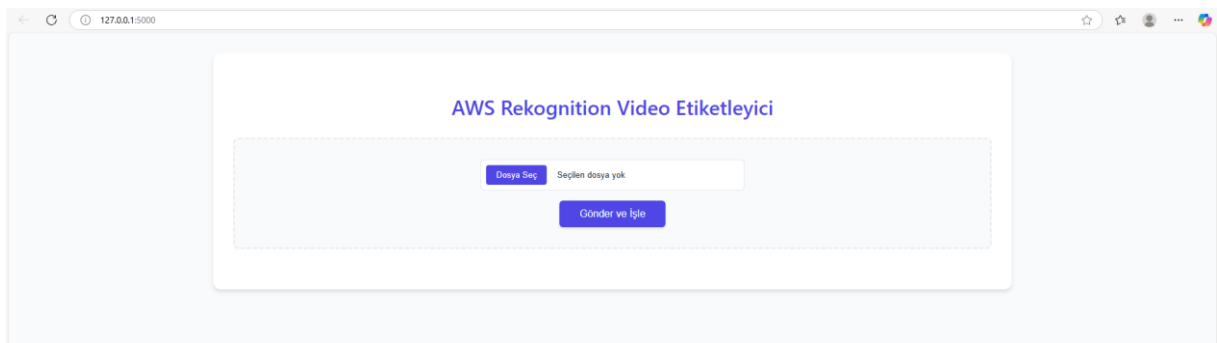
Uygulama sonlandığında kaynaklar temizlenir.

```
if __name__ == '__main__':  
    try:  
        app.run(  
            host='0.0.0.0',  
            port=5000,  
            debug=True,  
            use_reloader=False  
        )  
    except Exception as e:  
        print(f"Uygulama hatası: {e}")  
    finally:  
        cleanup()
```

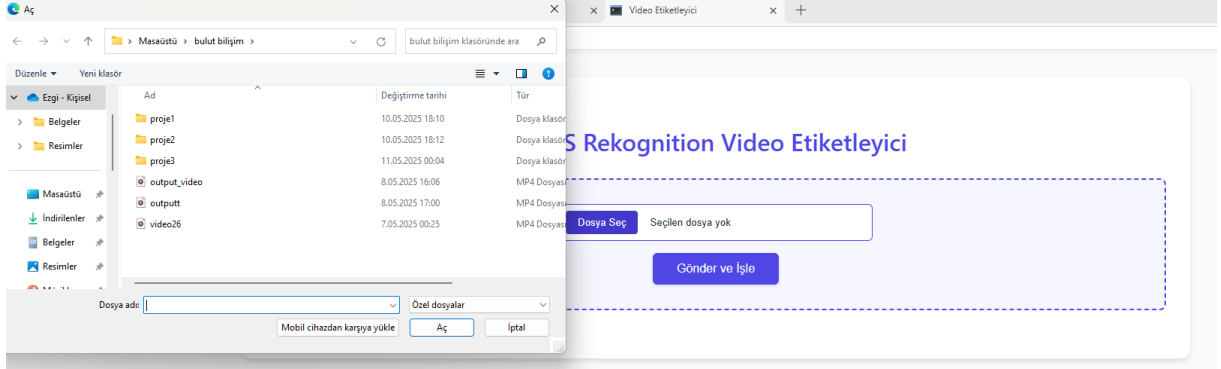
Uygulamanın Çalışması

app.py dosyası çalıştırılır. Terminaldeki <http://127.0.0.1:5000> adresine Ctrl +click ile erişilir.

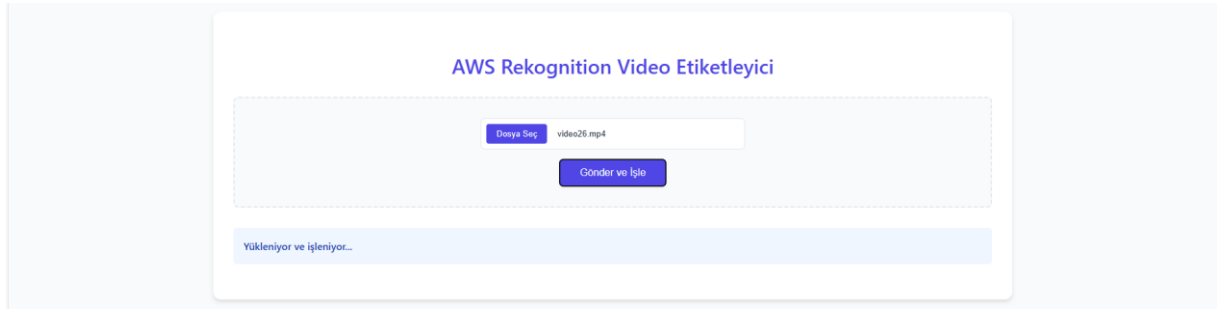
```
PS C:\Users\ezgis\Desktop\bulut bilişim\proje3> python -u "c:\Users\ezgis\Desktop\bulut bilişim\proje3\app.py"  
* Serving Flask app 'app'  
* Debug mode: on  
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.  
* Running on all addresses (0.0.0.0)  
* Running on http://127.0.0.1:5000  
* Running on http://192.168.1.3:5000  
Press CTRL+C to quit  
█
```



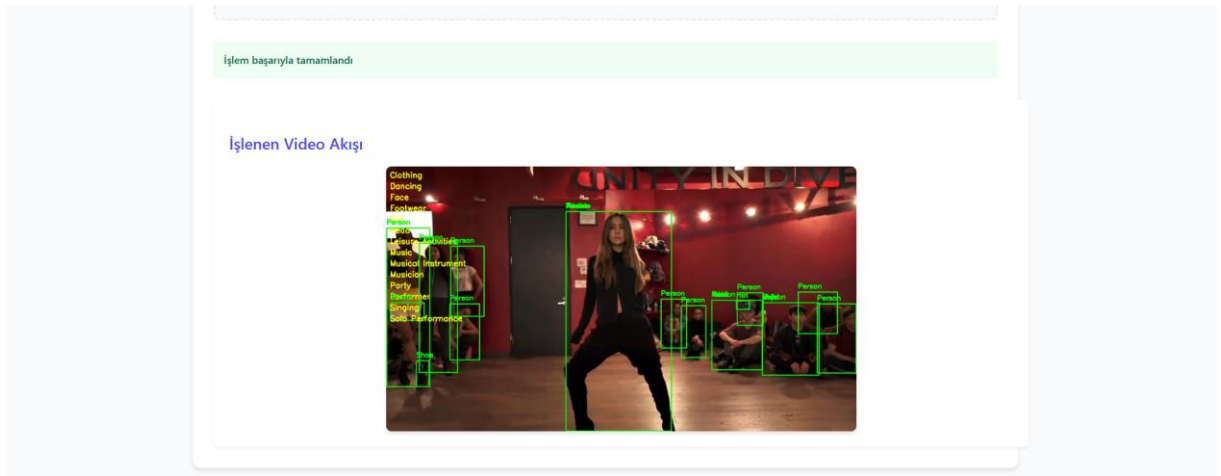
Dosya Seç butonundan dosya seçilir.



Gönder ve İşle butonuna basılıp videonun işlenmesi beklenir.



İşlenen video akışı ekranda gösterilir.



Ctrl+C ile uygulama kapatılır.

```
Press CTRL+C to quit
127.0.0.1 - - [12/May/2025 00:13:35] "GET / HTTP/1.1" 200 -
uploads\video26.mp4 dosyası video-akisi-bucket/video26.mp4 olarak yüklendi.
AWS Rekognition sonucu bekleniyor...
Bekleniyor... İşlem devam ediyor...
Bekleniyor... İşlem devam ediyor...
Bekleniyor... İşlem devam ediyor...
Bekleniyor... İşlem devam ediyor...
Bekleniyor... İşlem devam ediyor...
Bekleniyor... İşlem devam ediyor...
Bekleniyor... İşlem devam ediyor...
Etiketleme tamamlandı ✓
Etiketler uploads\video26_labels.json dosyasına kaydedildi.
127.0.0.1 - - [12/May/2025 00:16:10] "POST /upload HTTP/1.1" 200 -
127.0.0.1 - - [12/May/2025 00:16:10] "GET /video_feed HTTP/1.1" 200 -

Kaynaklar temizlendi, uygulama kapatılıyor...

Kaynaklar temizlendi, uygulama kapatılıyor...

Kaynaklar temizlendi, uygulama kapatılıyor...
PS C:\Users\ezgis\Desktop\bulut bilişim\proj3>
```

Oluşturduğumuz bucket’de uygulamaya yüklediğimiz video26.mp4 videosunu göreceğiz.

video-akisi-bucket [Info](#)

Objects	Metadata	Properties	Permissions	Metrics	Management	Access Points
Objects (8) Copy S3 URI Copy URL Download Open Delete Actions Create folder Upload						
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more						
<input type="text" value="Find objects by prefix"/>						
<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class	
<input type="checkbox"/>	video21.mp4	mp4	May 11, 2025, 00:18:44 (UTC+03:00)	8.1 MB	Standard	
<input type="checkbox"/>	video22.mp4	mp4	May 11, 2025, 00:41:26 (UTC+03:00)	8.1 MB	Standard	
<input type="checkbox"/>	video23.mp4	mp4	May 11, 2025, 00:49:10 (UTC+03:00)	8.1 MB	Standard	
<input type="checkbox"/>	video24.mp4	mp4	May 11, 2025, 14:48:00 (UTC+03:00)	8.1 MB	Standard	
<input type="checkbox"/>	video25.mp4	mp4	May 11, 2025, 15:36:06 (UTC+03:00)	8.1 MB	Standard	
<input type="checkbox"/>	video26.mp4	mp4	May 12, 2025, 00:15:20 (UTC+03:00)	8.1 MB	Standard	
<input type="checkbox"/>	video28.mp4	mp4	May 11, 2025, 00:24:50 (UTC+03:00)	13.7 MB	Standard	
<input type="checkbox"/>	video32.mp4	mp4	May 11, 2025, 00:30:36 (UTC+03:00)	5.4 MB	Standard	

Aws Rekognition’un video26.mp4 üzerinde yaptığı tespitleri Json formatında kaydetmiştik. Bu verileri MongoDB’ye kaydettik.

[+ Create Database](#)

Q Search Namespaces

projel2

projel2

sample_mflix

projel2.projel2

STORAGE SIZE: 804KB LOGICAL DATA SIZE: 2.04MB TOTAL DOCUMENTS: 7 INDEXES TOTAL SIZE: 34KB

[Find](#) [Indexes](#) [Schema Anti-Patterns](#) [Aggregation](#) [Search Indexes](#)

Generate queries from natural language in Compass

Filter

Type a query: { field: 'value' }

[Reset](#) [Apply](#) [Options](#)

QUERY RESULTS: 8-8 OF 8

```
{
  "_id": "68211397810608e1d6f8e7bb",
  "video_name": "video26.mp4",
  "analysis_date": "2025-05-12T00:16:07.738+00:00",
  "Labels": {
    "JobStatus": "SUCCEEDED",
    "VideoMetadata": {
      "Codec": "h264",
      "DurationMillis": 103480,
      "Format": "QuickTime / MOV",
      "FrameRate": 25,
      "FrameHeight": 720,
      "FrameWidth": 1280,
      "ColorRange": "LIMITED",
      "NextToken": "rhk9KX3Rnw7N0LTgp/Wbpctpy0h0AwZokwXBW4Bq8fpr7Kue3BRZLK3yWYJC+UR7xT/_",
      "Labels": [
        {
          "Timestamp": 40,
          "Label": {
            "Name": "Accessories"
          }
        }
      ]
    }
  }
}
```

[PREVIOUS](#) **8 of 8 results** [NEXT](#)

```
_id: ObjectId('68211397810608e1d6f8e7bb')
video_name: "video26.mp4"
analysis_date: 2025-05-12T00:16:07.738+00:00
Labels: Object
  JobStatus: "SUCCEEDED"
  VideoMetadata: Object
    Codec: "h264"
    DurationMillis: 103480
    Format: "QuickTime / MOV"
    FrameRate: 25
    FrameHeight: 720
    FrameWidth: 1280
    ColorRange: "LIMITED"
    NextToken: "RhKFWXK3Rnw7NOLTgp/WbpctpbyDhDAwIoW+X8M4Bqb8FpR7Kue3BRZLX3yWY3E+UR7xf/..."
  Labels: Array (1000)
    0: Object
      Timestamp: 40
      Label: Object
        Name: "Accessories"
```

